AMBiDDS: A system for Automatic Mining of Big Discrete Data-Sets

Ola Ågren
Department of Applied Physics and Electronics
Umeå University
SE-901 87 Umeå
Sweden
Email: ola.agren@umu.se

***Abstract*—This paper introduces an automatic algorithm that can be seen as an extension to the Eclat algorithm, as well as a corresponding proof of concept prototype. It uses inverted indices and statistical pruning of the possible solution space as early as possible.**

***Keywords*-Data mining, inverted indices, statistical pruning**

I. Introduction

Data mining denotes a number of different procedures and methods for finding interesting information in large amounts of data. Other names used for the concept include deductive learning, exploratory data analysis, and data driven discovery [1].

Most data mining tools will not directly work on the real time database that contains day to day operations of a business, but are often operating on a modified and/or summarised database called a *data warehouse* [2]–[4]. These databases usually contain summarised or otherwise aggregated information from the "live" databases, but the data tends to be cleaned, i.e., no false or extraneous data [2], [4].

Another difference between a standard database and a data mining system is in their operation. The user of a database system expects a crisp answer to each query (e.g. is a seat available on a certain flight). The data mining user might not know what he or she is looking for. The answer given by the data mining system might even be in the form of meta-data describing something in the database [2].

While there exist various types of data mining system operating on data warehouses, most fall in one of two classes:

- A *predictive* system will make some sort of a prediction for new values based on already known values in the database.
- A *descriptive* system tries to find relationships in data, e.g. patterns. Most of the time it is not used to predict future values, but can be used to analyse different attributes in the data.

The main focus of this work is on a specific version of the latter – finding *association rules* in a data warehouse. Mining for association rules is looking for patterns where one event is connected to another event. The rules found are usually in the form of $X \Rightarrow Y$, as given in Def. 2.1.

A. Unsupervised Mining for Association Rules

Unsupervised systems will automatically search for associations without being guided by input from the user. The most commonly used algorithm is called *Apriori* [2], [5].

The basic idea of *Apriori* is that only subsets of large sets can be large, and only large subsets can give new information that is potentially important. This means that the possible solution space can be pruned quickly while checking the combination of all itemsets that differ in only one member. A support parameter s is used in Apriori to decide which itemsets are considered large. This means that the system will ignore rules with high confidence if the support is too small [1], [2], [5].

B. Depth-First Based Approaches

A different approach is trying to find the association rules using a depth based version, such as with *Eclat* [6]. One of the major strengths of this approach is that most of the data (especially all terms in the potential rules) require very little memory. Only the data required for each recursion for the current path needs to be stored, everything else can be disposed of whenever we leave an item.

It has been shown that using a *DF* (Depth-First) algorithm works approximately as well as the Apriori algorithm, depending mostly upon the database being mined [7]. The algorithm described in this work belongs to this category, while using pruning (and gives output) based on statistically significant changes of the itemsets. It can also give similar output as Apriori and Eclat, as can be seen in Section III-A.

C. Other algorithms

Some other algorithms include:

- AprioriDP [8] uses Dynamic Programming in Frequent itemset mining. The working principle is to eliminate the candidate generation like frequent pattern-tree, but it stores support count in specialized data structure.
- CBPNARM [9] mines association rules using context variables on the basis of which the support of an itemset is changed on the basis of which the rules are finally populated to the rule set.
- Another approach is the Node-set-based algorithms, they use nodes in a coding frequent pattern-tree to represent

978-1-4673-9795-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSCI.2015.142

itemset, employing a depth-first search strategy to find frequent itemsesets using node set intersection. Three examples include FIN [10], PrePost [11] and PPV [12].

D. Paper Organization

The rest of the paper is organised as follows; Section II contains the definitions and algorithms, Section III describes the prototype, and Section IV discusses the results.

II. Method

Definition 2.1: Let $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ be a set of *items* or *itemset*. Let $\mathcal{D}$ be a set of *transactions*, where each transaction T is a set of items, $T \subseteq \mathcal{I}$. We say that a transaction T *contains* X if $X \in T$. The fraction of transactions containing X is called the *frequency* of X. An *association rule* is an implication in the form $Y \Rightarrow Z$, where $Y, Z \subseteq \mathcal{I}$, and $Y \cap Z = \varnothing$. This rule holds in the transaction set $\mathcal{D}$ with a *confidence* α if the fraction of the transactions containing Y that also contain Z is at least α. The rule has *support* s in the transaction set $\mathcal{D}$ if the fraction of the transactions in $\mathcal{D}$ that contain $Y \cap Z$ is at least s [2].

Definition 2.2: A *path* is a conjunction of the transactions by all the items contained in it, so path = $\bigwedge$item.

Definition 2.3: By using the standard transformation from the binomial to normal form and given that p is the relative frequency, α equals the interval and n is the number of records we can say that the *confidence interval* is given by Eq. (1).

$$\mathrm{CI} = p \pm \alpha \sqrt{\frac{p(1-p)}{n}} \tag{1}$$

Corollary 2.1: Def. 2.3 breaks down when $np < 5$ or $n(1-p) < 5$, so we have a logical point of pruning the solution space if either of these cases appear.

Definition 2.4: A *statistically significant change* is when $\mathrm{P}(\mathrm{item}_1|\mathrm{path}\wedge\mathrm{item}_2)$ is not in the confidence interval for $\mathrm{P}(\mathrm{item}_1|\mathrm{path})$.

A. The Algorithm

It is basically a depth first search, stopping at each node just for pruning the solution space as well as presenting any changes from the parent node. All subscripts denote index positions and superscripts denote iteration positions.

The line marked with † has not been implemented in our prototype, it uses numerical order instead. The lines marked with ‡ here and in Section II-B are ignored when doing a full resulting set in the prototype.

Globals:

	Order	All items, in cardinality order †
	CI_i^j	confidence interval for all items
	PATH	array of items

Parameters:

	c	cutoff value
	α	confidence interval multiplier

CurrSpace^0 $\leftarrow$ all transactions
Card^0 $\leftarrow$ |CurrSpace|
Ignore^0 $\leftarrow \emptyset$
for each $i \in$items
  item$_i^0$ = read inverted index$_i$
  if (Card$^l - c$) $\geqslant$ |item$_i^0$| $\geqslant c$ **then**
    $p = |\mathrm{item}_i^0|/\mathrm{Card}^0$
    $\mathrm{CI}_i^0 = p \pm \alpha\sqrt{p(1-p)/\mathrm{Card}^0}$
  else
    Ignore^0 $\leftarrow$ Ignore$^0 \cup \{i\}$
for each $i \in$ Order - Ignore^0
  AmbiRec(1, i, item$_i^0$, Ignore$^0 \cup \{i\}$)
  Ignore^0 $\leftarrow$ Ignore$^0 \cup \{i\}$ ‡

B. AmbiRec

Parameters:

	l	current level
	k	current item
	C	current transaction space
	Ignore^l	current list of ignores

PATH(l) $\leftarrow k$
Card^l $\leftarrow |C|$
 # Present data that has changed from parent
for each $i \in$ Order - Ignore^l
  item$_i^l$ $\leftarrow$ item$_i^{l-1} \cap C$
  $p = |\mathrm{item}_i^l|/\mathrm{Card}^l$
  if $p \notin \mathrm{CI}_i^{l-1}$ **then**
    Present stat. significant change
  if (Card$^l - c$) $\geqslant$ |item$_i^l$| $\geqslant c$ **then**
    $\mathrm{CI}_i^l = p \pm \alpha\sqrt{p(1-p)/\mathrm{Card}^l}$
  else
    Ignore^l $\leftarrow$ Ignore$^l \cup \{i\}$
 # Handle recursion
for each $i \in$ Order - Ignore^l
  AmbiRec(l+1, i, item$_i^l$, Ignore$^l \cup \{i\}$)
  Ignore^0 $\leftarrow$ Ignore$^0 \cup \{i\}$ ‡

III. Results

The prototype is written in ANSI C89, using the BitSet [13] library to handle all the set operations performed on the the inverted indices.

The number of visited items and the time required from the two major testing databases can be seen in Table I. It shows that time required (tested on a laptop with a 2.00GHz i7-3537U processor) is close to linear to the total number of items visited, which is the expected result. Furthermore, most of the

changes are indeed statistically significant when changing the path: A minimum of 64.58% of the normal path changes are statistically significant in all of our test databases, while all of the changes to a confidence of 0% or 100% are significant.

TABLE I
NUMBER OF ITEMS VISITED AT A SPECIFIC DEPTH AND THE TIME REQUIRED USING THE TWO MAJOR TEST DATABASES

	Small database		Large database	
Depth	***Normal***	***Full***	***Normal***	***Full***
1	102	102	222	222
2	304	405	2937	5398
3	366	709	7189	29360
4	259	777	6542	70752
5	83	283	2820	84729
6			573	50845
7			46	13325
8				864
Time	0.64s	1.31s	19.96s	4m 14.25s

A. Association Rules

Adding the flag to get association rules generates data that corresponds to what you get from Eclat or Apriori. There is also a small perl script (called *asso*) to rewrite the output from the prototype to just the set of associations with the corresponding support and confidence (see Fig. 1 for a short example).

IV. DISCUSSION

The major novelty in this paper is the use of the inverted indices directly in the algorithm and the statistical pruning. The general idea for the latter is that if item A and item B are *statistically independent* then $P(A \cup B) = P(A)P(B)$. Any major change when applying one more step in a path indicates that there is a statistical dependance (positive or negative) or possibly even a mutually exclusive situation between them. This can easily be used to (dis)prove correlations between items.

Generating all commonly generated association rules to the statistically generated rules was an added bonus, meaning that the results in general matched the output of state of art algorithms as well.

The only major drawback of the prototype is that it currently does not handle the items in decreasing cardinality order, meaning that it might at times start with one node and never go "backwards" in the order. Switching off the tree pruning removes this, but at a much higher computational cost as well as bigger output files. As an example, *text* $\wedge$ *c* $\Rightarrow$ *program* is given by the program but not *text* $\wedge$ *program* $\Rightarrow$ *c*. In this particular case they do have the same support, but very different confidences (68.52% vs. 99.77%). The latter case is close enough as to be pruned by the algorithm (due to Cor. 2.1). There are currently plans for an updated prototype that would use the sorted order instead.

```
Total = 12492
ascii => text [12.4959974383606, 100.0]
text => english [7.356708, 10.052505]
text => c [10.566763, 14.438854]
text => program [15.385847, 21.023846]
text => data [0.184118, 0.251586]
text => bourne [0.672430, 0.918836]
text => shell [1.304835, 1.782980]
text => script [1.360871, 1.859549]
text => html [22.214217, 30.354408]
```

Fig. 1. The first ten lines from *asso* when applying it to the output of the prototype

REFERENCES

[1] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*. San Francisco, California: Morgan Kaufmann Publishers, Inc., 2001.

[2] M. H. Dunham, *Data Mining, Introductory and Advanced Topics*. Englewood Cliffs, New Jersey: Prentice Hall, inc., 2003.

[3] M. Humphries, M. W. Hawkins, and M. C. Dy, *Data Warehousing: Architecture and Implementation*. Upper Saddle River, New Jersey: Prentice Hall PTC, 1999.

[4] Two Crows Corporation, "Introduction to Data Mining and Knowledge Discovery, Third Edition," Potomac, MD, USA, 2005.

[5] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Reading, Massachusetts: Addison-Wesley, 2006.

[6] M. J. Zaki, "Scalable Algorithms for Association Mining," *IEEE Transactions on Knowledge & Data Engineering*, vol. 12, no. 03, pp. 372–390, May/Jun. 2000.

[7] J. Hipp, U. Güntzer, and G. Nakhaeizadeh, "Algorithms for Association Rule Mining – a General Survey and Comparison," *SIGKDD Explor. Newsl.*, vol. 2, no. 1, pp. 58–64, Jun. 2000. [Online]. Available: http://doi.acm.org/10.1145/360402.360421

[8] D. Bhalodiya, K. Patel, and C. Patel, "An efficient way to find frequent pattern with dynamic programming approach," in *Proceedings of the* 4^{th} *Nirma University International Conference on Engineering (NUiCONE)*, Ahmedabad, India, Nov. 28-30, 2013.

[9] M. Shaheena, M. Shahbazb, and A. Guergachic, "Context Based Positive and Negative Spatio-Temporal Association Rule Mining," *Knowledge-Based Systems*, vol. 37, pp. 261–273, Jan. 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950705112002237

[10] Z. Deng and S.-L. Lv., "Fast mining frequent itemsets using Nodesets," *Expert Systems with Applications*, vol. 41, no. 10, pp. 4505–4512, Aug. 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417414000463

[11] Z. Deng, W. ZhongHui, and J. Jiang, "A new algorithm for fast mining frequent itemsets using N-lists," *SCIENCE CHINA Information Sciences*, vol. 55, no. 9, p. 2008, 2012. [Online]. Available: http://info.scichina.com/sciFe/EN/abstract/article_508369.shtml

[12] Z. Deng and W. ZhongHui, "A New Fast Vertical Method for Mining Frequent Patterns," *International Journal of Computational Intelligence Systems*, vol. 3, no. 6, pp. 733–744, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417414000463

[13] O. Ågren, "BITSET: Implementing Sets of Natural Numbers Using Packed Bits," Umeå University, Umeå, Sweden, Tech. Rep., Oct. 2002, uMINF 02.10, ISSN 0348-0542.

[14] ——, "Automatic Generation of Concept Hierarchies for a Discrete Data Mining System," in *Proceedings of the International Conference on Information and Knowledge Engineering (IKE '02)*, Las Vegas, Nevada, USA, Jun. 24-27, 2002, pp. 287–293.

[15] ——, "CHIC: A Fast Concept HIerarchy Constructor for Discrete or Mixed Mode Databases," in *Proceedings of the Fifteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'03)*, San Francisco, California, USA, Jul. 1-3, 2003, pp. 250–258.

APPENDIX

Users' Guide to the prototype

A. Input Data

It uses the same type of databases as did the CHiC concept hierarchy constructor [14], [15], i.e. a directory for each database containing the following files:

- DB — is a list of the discrete items in the database, one on each line.
- names — is a list of the transaction names (not used in this work).
- stdin — is a hex-coded version of the database, one transaction per line. Only used to calculate the number of transaction in the database in the prototype.
- 400– — are the inverted indices, one per discrete item. The file names are given as a hexadecimal enumeration, starting with $400_{16} = 1024_{10}$. Each line in the file contain a transaction number, given in hexadecimal form.

B. Parameters and output from the prototype

Each line of output is given using tab characters as separators between each major data point. They start out with a hexadecimal line number, followed by a number that corresponds to the search depth. In all cases, an integer value is given in hexadecimal form, while real numbers are given in decimal form.

The basic output lines are as follows:

- F — indicates that the two items given share the same transactions, and are therefore handled using the first one. Only used in the header, at depth 0.
- START ... STOP — indicates that we are adding a "$\wedge$ item" at the end of the PATH. On the first line, first the item and the cardinality of it is given. On the last line, there is a reference to the line number of the first line.
- PATH — is a conjunction of each given item. Please note that this list is space separated.
- ZR(S) — the given items do not appear in any remaining transactions in the path. The S is added if the change from the previous path is statistically significant.
- ON(S)(T) — the given item appears in all remaining transactions in the path. The *S* is added if the change from the previous path is statistically significant and *T* indicates that it is a terminal.
- CHNG — starts information about a item, given the current PATH.
- (C)CONF — contains the confidence interval for the last given CHNG line. The *C* correspond to a change that is statistically significant.

1) Mining for association rules: Adding the *-a* flag indicates that actual association rules should be mined for, so the prototype will give output more comparable to that of Apriori.

The following two line type are added to the output:

- TOT — with a single number, corresponding to the number of transactions in the database.
- ASS — is the association for the latest CHNG line, given from the latest PATH: PATH $\Rightarrow$ CHNG. The two numbers given correspond to the support and confidence of this rule, given as percentages.

2) Other command line parameters: The *-c <value>* flag can be used to set the cutoff (c in Section II-A). The default value is 5, and can only be increased.

The *-d* flag is mainly for debugging purposes, creates a very verbose output with a lot more answer types. Please note that this output is subject to change between versions of the software.

The *-D <path>* flag is used to point out the database to use.

The *-i <value>* flag is used to change the confidence interval size, the default value is 1.96 for a 95% confidence interval.

The *-M <value>* flag can be used to set a maximum cutoff depth for the prototype.

The *-f* flag is used to get a richer data mining, i.e. it will try to go combinatorially through all possible paths. This includes possibly reaching the same logical path multiple times combinatorially by going first $A \rightarrow B$ and then $B \rightarrow A$. It will find all possible search paths in the dataset, but it is much slower.

There is also a flag (*-h*) to get output that is more easily parsed by humans. This means that the search depth is shown as a number of tab stops instead, and the references to items are given as the actual items rather then the corresponding hexadecimal values.